Reliable Web Service Publication and Discovery through Model-Based Testing and Verification

Ervin Ramollari[1], Dimitris Dranidis[2], Anthony J. H. Simons[3]

[1]SEERC, 17 Mitropoleos Str, Thessaloniki, Greece, sname@seerc.org

[2]City College, Dept. of Computer Science, 13 Tsimiski Str, 54624 Thessaloniki, Greece, dranidis@city.academic.gr

[3]The University of Sheffield, Dept. of Computer Science, Western Bank, Sheffield, S10 2TN, UK,a.simons@sheffield.ac.uk

Currently, the issues of trust and dependability on third-party Web services are becoming key factors to the adoption of service oriented computing in industrial environments. As a result, robust service testing and verification techniques are highly important in order for consumers to build confidence on third-party Web services. In this paper we propose modelling the behaviour of a Web service using stream X-machines, in order to derive a complete test set and to perform model-based testing. We apply these techniques in a novel publication and discovery approach involving all three main actors in a SOA environment, i.e. the service provider, the service broker, and the service consumer. The provider augments the service interface description (WSDL) with a stream X-machine (SXM) model reflecting the Web service behaviour. This model is both utilised by the broker during publication to derive a test set and verify Web service behavioural conformance, and by the consumer during discovery to perform service selection based on model validation.

Keywords
model-based testing, SOA, stream X-machines, validation, Web services.

1. Introduction

Service Oriented Computing is a new computing paradigm that utilizes services as the key abstraction to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments [1]. Services are loosely coupled, reusable, and implementation-independent software modules with well-defined interfaces. They can be *described*, *published*, *discovered*, and dynamically *assembled* for developing massively distributed, interoperable, evolvable systems.

Services are made available by service providers within or outside the boundaries of an enterprise, and invoked by service consumers.

Currently, the prevailing alternative to implement a Service-Oriented Architecture (SOA) is the Web services framework, which is founded on widely accepted standards, such as WSDL for the service interface description, SOAP for the communication protocol, and UDDI for service discovery. Interaction between the three main parties that are involved, that is, service consumers, service providers, and service brokers, occurs as follows: service consumers discover Web services in a UDDI service registry maintained by service brokers. They retrieve WSDL descriptions of Web services offered by service providers, who previously published those WSDL descriptions in the UDDI registry. After the WSDL has been retrieved, the service consumer binds to the service providers by invoking the service through SOAP.

Given the increasing number of Web services that are being offered by third-party providers, the issues of *trust* and *dependability* on Web services are becoming increasingly important and considered as key factors to the adoption of service oriented computing in industry. As a result, *robust service testing, verification, and validation techniques are crucial* in order for consumers and integrators to build confidence on third-party Web services. In other words, service consumers need to ensure that advertised Web services are what they need, and that their implementations have been verified. However, one major obstacle in achieving these goals is that the WSDL standard lacks support for capturing the semantics relating to functional and non-functional aspects of a service. Therefore, it is not possible to guarantee that a discovered service advertisement matches a service request in all respects, and this may lead to inappropriate bindings.

One of the important aspects that the WSDL specification lacks is Web service *behaviour*, i.e. the definition of preconditions and effects of each Web service operation as well as the implied sequencing of these operations. Behavioural specifications are especially useful in cases of Web services assuming an interaction protocol (stateful Web services) and Web services operating on persistent data. In this paper we propose an approach which tries to fill this gap by introducing formal specifications of Web service behaviour to its interface description. For this purpose we choose stream X-machines (SXMs) [2], which we consider an intuitive and powerful formalism. Not only do SXMs allow for unambiguous specification of Web service behaviour, but they are also highly useful to perform model-based testing of a Web service implementation under test (IUT). Research on stream X-machine offers a test generation method, which under certain assumptions is proven to find all faults in the implementation [3, 4]. Our approach makes use of these benefits requiring the cooperation of all three main stakeholders in a SOA environment (SOA triangle), i.e. the service provider, the service broker, and the service consumer. The provider's role is to create a SXM model reflecting the behaviour of the provided Web service implementation, and add it to WSDL during the publication process. Based on this model, the broker is able to derive the necessary test cases, which are run in order to verify behavioural equivalence between the advertised model and the implementation. Only services with successful test results are accepted (i.e. checked-in) in the registry. On the other hand, during the discovery process, the consumer is provided with a number of service candidates fulfilling the request. Through the provided SXM models, the consumer can validate the behaviour of candidate services against consumer needs, a process that aids in the

selection of the most appropriate service. Therefore, our approach ensures that clients bind with Web services providing a suitable behaviour, and a verified implementation.

The rest of this paper is structured as follows. Section 2 presents a summary of related work in model-based testing of Web services and in related publication/discovery approaches. Section 3 provides a description of the used stream X-machine formalism, the Web service modelling process, and the associated complete functional testing method, illustrated with a simplified shopping cart case study. Section 4 provides an overview of the approach for reliable Web service publication and discovery based on stream X-machines, described from the perspectives of the service provider, the service broker, and the service consumer. In the end, Section 5 concludes the paper by summarizing the main points of the presented work, and suggesting directions for future work necessary to realise the described approach.

2. Related Work

A number of approaches have been proposed for applying model-based testing to verify Web services. The authors in [5] propose an algorithm, which translates Web service descriptions annotated in WSDL-S into an equivalent Extended Finite State Machine representation, which extends simple Finite State Machines with the addition of a multi-dimensional structure (memory) and the modification of the state transition function so that it maps a (input, initial state, initial memory) tuple to a (output, new state, new memory) tuple. The WSDL-S document is enhanced with references to OWL-S concepts, as well as to SWRL rules, which explicate the behaviour of individual Web service operations in the form of inputs, outputs, preconditions, and effects (IOPE). The resulting EFSM model, consisting of a *single* state, is then exploited, using *any* appropriate test case generation algorithm, to derive an effective test set for verifying a Web service behavioural conformance.

Keum et al [6] present another model-based testing approach using Extended Finite State Machines, which extend finite state machines with memory, and with computing blocks and predicate conditions for state transitions. A procedure is described for semi-automatically deriving the EFSM model from a WSDL specification and additional user input. The model covers behavioural aspects of *stateful* Web services, and the resulting test cases represent sequences of invocations of Web service operations. The authors provide experimental results showing that their method has the potential to find more faults compared to other methods, but notably, with a resulting test case set that is much larger and takes more time to execute.

Some other works have proposed the application of model-based verification of Web services in the context of more complete approaches. Bertolino et al [7] describe an envisaged registry-based Web service verification framework. The provider augments the WSDL document with behavioural descriptions in a UML 2.0 Protocol State Machine (PSM) diagram, which is then translated to a Symbolic Transition System (STS). On the other hand, the broker utilises the attached STS model to automatically generate the test cases and run them on the provided Web service for behavioural conformance verification. Upon successful test results the Web service is published in the UDDI registry as a certified service. For this reason, the authors call their approach

an "Audition" framework, where the Web service undergoes a monitored trial before being put "to stage".

Heckel and Mariani [8] use graph transformation rules to model the behaviour of Web service operations and apply them in a reliable Web service publication and discovery approach. Both the behaviour of the advertised service by the service provider *and* the requested service by the service consumer need to be modelled in terms of graph transformation rules. A test case derivation method is employed to verify that the actual service implementation conforms to the provided model. This verification is performed by the service broker before services are accepted in the registry, resulting in what the authors refer to as high-quality service discovery agencies. In addition, during discovery, the service broker enables matchmaking of request and advertisement models that are expressed as graph transformation rules, in order to return service candidates satisfying the consumer's behavioural constraints.

In the approach we propose we find some advantages relative to the aforementioned approaches. The employed X-machine complete functional testing method can be proven to reveal all implementation errors, under certain design-for test conditions that the model and the implementation have to satisfy [3, 4]. Additionally, in contrast to the approach proposed by Heckel and Mariani, this approach does not require models of both the service request and service advertisement during service discovery, since service selection is performed through behavioural validation at the consumer site. Indeed, it is impractical to assume that the consumer knows in advance the detailed behaviour of the requested Web service and can create a formal model of that behaviour.

3. Modelling Web Service Behaviour with Stream X-Machines

3.1 Stream X-machines

Stream X-machines (SXMs) [2] is a computational model capable of modelling both the data and the control of a system. SXMs are special instances of X-machines introduced by Eilenberg [9]. They employ a diagrammatic approach of modelling the control by extending the expressive power of finite state machines. In contrast to finite state machines, SXMs are capable of modelling non-trivial data structures by employing a memory, which is attached to the state machine. Additionally, transitions between states are not labelled with simple input symbols but with processing functions. Processing functions represent internal system transitions triggered by input symbols under specific memory conditions, and produce output symbols while modifying the memory. The benefit of the addition of a memory structure is that state explosion is avoided and the number of states is reduced to those states which are considered critical for the correct modelling of the system's abstract control structure. A divide-and-conquer approach to design allows the model to hide some of the complexity in the transition functions, which are later exposed as simpler SXMs at the next level.

A stream X-machine is defined as an 8-tuple, $(\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:

- Σ and Γ is the input and output finite alphabet respectively;
- Q is the finite set of states;
- M is the (possibly) infinite set called memory;
- Φ, which is called the type of the machine SXM, is a finite set of partial functions (processing functions) φ that map an input and a memory state to an output and a new memory state, $\varphi: \Sigma \times M \rightarrow \Gamma \times M$;
- F is the next state partial function that given a state and a function from the type Φ, provides the next state, $F: Q \times \Phi \rightarrow Q$ (F is often described as a transition state diagram);
- q_0 and m_0 are the initial state and memory respectively.

The sequence of transitions (path) triggered by the stream of input symbols is called a *computation*. The computation halts when all input symbols are consumed. The result of a computation is the sequence of outputs symbols produced by this path.

Apart from being formal as well as proven to possess the computational power of Turing machines [3], SXMs have the significant advantage of offering a testing method [3, 4] that ensures conformance of an implementation to a specification. This method generates test sets for a system specified as a SXM whose application ensures that the system behaviour is identical to that of the specification provided that the system is made of fault-free components and some explicit design-for-test requirements are met.

In order to allow for specifications of stream X-machines, the XMDL (X-Machine Definition Language) language was introduced in [10] and fully developed in Kefalas [11]. XMDL serves as an interlingua for the development of tools supporting Stream X-machines [12]. An extension of XMDL to support an object-based notation was suggested in [13]. The object-based extension, called XMDL-O, enables an easier and more readable specification of Stream X-machines and is employed in this paper for the specification of the example Web service.

3.2 Shopping cart example

We illustrate our modelling and test set generation method with a simplified version of a Web service that is intended to provide the backend functionality of a shopping cart to consumers, also described in a previous paper [14]. Similar Web services are already being used and made available over the Web, such as the Amazon Shopping Cart Web service (http://developer.amazonwebservices.com). The `ShoppingCart` Web service provides the following operations:

- The `login` operation allows authentication for using the service. It is invoked with the input message `LoginRequest` consisting of the username and the password of the user. The request message is represented as `LoginRequest(user, pwd)`. The operation sends back the response message `LoginResponse(result)`, where result is a boolean value; true indicates successful authentication.

- The `addToCart` operation adds an item to the shopping cart. It is invoked with the input message `AddToCartRequest` consisting of the identifier of the item to be added. The request message is represented as `AddToCartRequest(itemId)`. The

operation sends back the response message `AddToCartResponse(itemId)`. It is assumed that all item identifiers are valid and correspond to products that may be purchased.

- The `clearCart` operation removes all items from the shopping cart. It is invoked with the simple request message `ClearCartRequest` represented as `ClearCartRequest()` and it sends back the response message `ClearCartResponse()`.

- The operation `checkout` completes the shopping process. It is invoked with the simple request message `CheckoutRequest` represented as `CheckoutRequest()` and it sends back the response message `CheckoutResponse()`.

The ShoppingCart service is an example of a *stateful Web service*. This implies that the availability of Web service operations depends not only on the input, but also on the internal state of the service, which in turn results from previous operation invocations. For instance, the client is not allowed to perform any operation before authenticating, and checking out only makes sense with a non-empty cart. Viewed from another angle, stateful Web services are a form of conversational Web services, which assume an interaction protocol defining rules for a suitable sequencing of operation invocations. Understanding and verifying this interaction protocol is crucial for the interoperability between a consumer system and a provided Web service. In addition, the ShoppingCart service operates on *persistent data* (in contrast to a *transformational* Web service, which accepts some user input and returns a result). The presence of persistent data implies that the result of invoking a Web service depends on the state of the persistent data (such as the existence of a user account in an accounts database), in addition to the user input (such as login data). Stream X-machines are suitable to formally model both the behaviour of single operations (pre-conditions and effects on persistent data), as well as the expected sequencing of operations for successful interaction.

Some parallels can be drawn between a stateful Web service and a stream X-machine, given that both accept inputs and produce outputs, while performing specific actions and moving between internal states. SXM inputs correspond to request messages, outputs correspond to response messages, and processing functions correspond to operation invocations in distinct contexts. In addition, the service provider has to define the memory structure, not only as a substitute for internal state, but also to supply genuine test data that can become part of the generated test sequences. Figure 1 is the diagrammatical representation of the stream X-machine model of the ShoppingCart service. It has to be noted that the transitions on the diagram do not correspond to operations or messages of the Web service but to processing functions as defined later on. Furthermore, some transitions that represent exceptional behaviour are not shown in the diagram for the sake of clarity. For instance, attempting to invoke the operation `addItem` while the service is found at state waiting, will exercise the self-transition `faultyAddItem`. Similar transitions exist for the rest of the operations and the states.

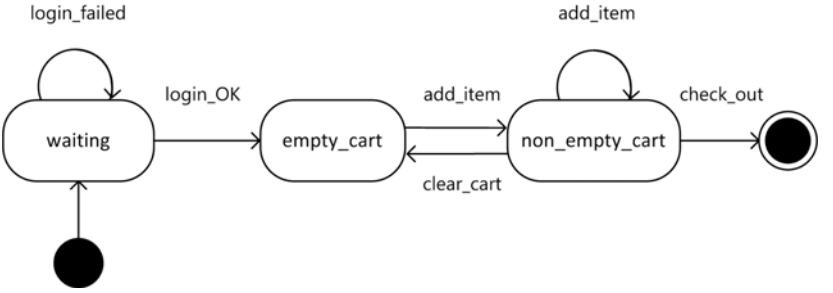

Figure 1 Stream X-machine model of the shopping cart Web service example

The memory in the ShoppingCart service example is used to store information about valid user accounts and the contents of the shopping cart. The following XMDL-O code shows the definition of accounts as a set of Account objects and the cart as a set of item identifiers (strings). For the purpose of testing the system we assume that there are two valid user accounts.

```
#class Account {
     username: string,
     password: string,
}.

#objects:
     account1: Account,
     account2: Account,
     accounts: set_of Account,
     cart: set_of string.

#init_values:
     account1.username <- "usr1",
     account1.password <- "pwd1",
     account2.username <- "usr2",
     account2.password <- "pwd2",
     accounts <- {account1, account2},
     cart <- emptySet.
```

State transitions in SXMs are labelled with processing functions. A processing function is triggered by an input event, when a specified guard condition holds, produces some output, and potentially updates (modifies) the memory. The updating of the memory consists of a sequence of assignments as specified in the update part of the processing function definition. The following XMDL-O code shows the definition of processing functions. When modelling Web services, the inputs and the outputs of the processing functions correspond intuitively to request and response messages of Web services respectively.

```
#fun loginOK( LoginRequest(?usr, ?pwd) ) =
     if ?account \= null and ?pwd = ?account.password
     then ( LoginResponse(true) )
     where
```

```
        ?account <- head (select(username = ?usr, accounts)).

#fun loginFailed( LoginRequest(?usr, ?pwd) ) =
     if ?account = null or ?pwd \= ?account.password
     then ( LoginResponse(false) )
     where
         ?account <- head (select(username = ?usr, accounts)).

#fun addItem( AddToCartRequest(itemId) ) =
     then ( AddToCartResponse() )
     update
         cart <- itemId addsetelement cart.

#fun clear( ClearCartRequest() ) =
     then ( ClearCartResponse() )
     update
         cart <- emptySet.

#fun checkOut( CheckOutRequest() ) =
     if cart \= emptySet
     then ( CheckOutResponse() ).
```

3.3 Test set derivation

A main strength of modelling systems with SXMs is the existence of a test generation method which under certain assumptions [3, 4], is proven to find all faults in the implementation. Examples of faults that can be detected in the implementation include erroneous transition labels, erroneous next-states, missing states, extra states, etc [15]. The testing method is a generalization of the W-method [16]. It works on the assumption that the system specification and the implementation can be both represented as stream X-machines with the same type (i.e. both specification and implementation have the same processing functions) and satisfies the following design for test conditions: completeness with respect to memory (all processing functions can be exercised from any memory value using appropriate inputs) and output distinguishability (any two different processing functions will produce different outputs if applied on the same memory/input pair).

When the above requirements are met, the Stream X-machine testing method may be employed to produce a complete test set of input sequences which can be used for the verification of the implementation under test. In fact it is proved that only if the specification and the implementation are behaviourally equivalent, the test set produces identical results when applied to both of them. Otherwise it is guaranteed that it will reveal the faults in the implementation.

The first step to constructing the test set of input sequences is based on the application of the W-method to the associated finite state automaton of the SXM, by considering processing functions as simple inputs. The test set X for the associated automaton consists of sequences of processing functions and it is given by the formula:

$$X = S(\Phi^{k+1} \cup \Phi^k \cup \ldots \cup \Phi \cup \{\square\})W$$

Where W is a characterization set, S a state cover of the associated finite state automaton, and k is the estimated difference of states between the implementation and the specification. A characterization set is a set of sequences of processing functions for which any two distinct states of the machine are distinguishable and a state cover is a set of sequences of processing functions such that all states are reachable from the initial state. The W and S sets in the ShoppingCart Web service example are:

W = {<hloginOK>, <addItem>, <checkout>}
S = {<□>, <loginOK>, <loginOK, addItem>, <loginOK, addItem, checkout>}

The derived test set X, e.g. for $k = 0$, is the following (note that it is not completely presented):

X = {<loginOK>, <addItem>, <checkOut>, <loginOK, loginOK>, <loginFailed, loginOK>, <addItem, loginOK>, <clearCart, loginOK>, <checkOut, loginOK>, <loginOK, addItem>, <loginOK, checkOut>, <loginOK, loginOK, loginOK>, <loginOK, loginFailed, loginOK>, <loginOK, addItem, loginOK>, <loginOK, clearCart, loginOK>, <loginOK, checkOut, loginOK>, <loginOK, loginOK, addItem>, <loginOK, loginFailed, addItem>, <loginOK, addItem, addItem>, <loginOK, clearCart, addItem>, <loginOK, checkOut, addItem> …}

The above test-set X consists of sequences of operations. These sequences have to be converted to sequences of inputs. This is achieved by the fundamental test function as described in [3]. For instance, the sequence of operations <loginOK, addItem, addItem> is converted to the following sequence of inputs:

```
<loginRequest("usr1, "pwd1"),
addToCartRequest("912"),
addToCartRequest("875")>
```

To complete the process of test set generation and enable a testing engine to execute the test cases, these abstract test cases have to be mapped to executable test cases that the testing engine can understand.

4. Reliable Web Service Discovery and Publication Approach

The method described in the previous section for modelling and model-based testing of Web services using stream X-machines has applicability in a range of scenarios involving various stakeholders. In this section we describe the use of stream X-machine formal models in a reliable Web service publication and discovery approach [17, 18]. The approach is founded on the idea that augmenting Web service interface descriptions (WSDL) with formal behavioural specifications is beneficial in registry-based testing of provided Web services during publication, and in service selection by the consumer during discovery.

Figure 2 provides an overview of the proposed approach, which requires the cooperation of the three main stakeholders in a SOA environment: the service provider, the service broker, and the service consumer. The following subsections describe the

steps involved in the approach from the perspectives of each of these three stakeholders.

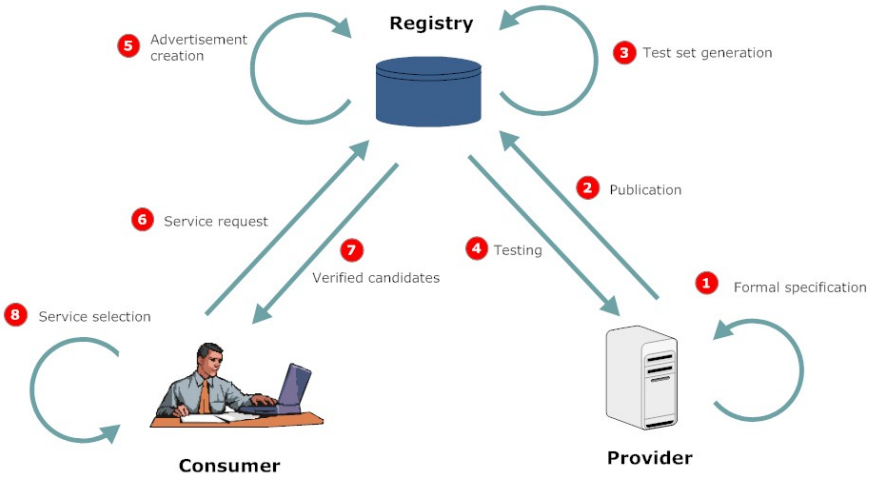

Figure 3 Overview of the publication and discovery approach

4.1 Provider's perspective

The service provider goes through data-level and behavioural-level analysis to derive a formal model reflecting the behaviour of the Web service that is to be published, using the stream X-machine (SXM) formalism [2]. The SXM model, expressed in a markup language such as XMDL [11], is then linked to the WSDL document of the Web service. Practically, this may be achieved by adding an SAWSDL annotation [19] that references the URI of the SXM markup document. The next step by the provider is the publication of the Web service to a service registry maintained by a broker. The publication query, which references the semantically annotated WSDL document at the provider site, initiates the publication procedure at the broker site.

4.2 Broker's perspective

A key role of the service broker in this approach is to verify the behaviour of the provided Web service implementation through model-based testing, and upon successful test results, to accept it in the service registry. This step is necessary to ensure that the implementation of the provided Web service really conforms to the advertised behavioural specifications. It is possible that this might not be the case, either because of insufficient testing at the provider site, or because of malicious intent. With the attached SXM specification, the broker is able to derive the test sequences for verification automatically. The theory of complete functional testing from X-machines

offers a method for deriving a complete, finite set of test cases, which is proven to find all faults in the implementation under test [3].

The input sequences and the expected output sequences produced by the testing algorithm are at the same level of abstraction as the stream X-machine model, so they need to be mapped to concrete data types (XSD), which can be understood by the Web service. This is possible if the provider uses a mechanism to link the abstract types in the XMDL model with the XSD types in the WSDL document. The test cases are then written in a form of executable tests, which are interpreted and run by a testing engine that communicates with the Web service under test through SOAP messages. If the test results are successful, i.e. the expected and produced outputs match, then the Web service implementation has been shown to be free of faults with respect to the behavioural specifications. In such a case, an advertisement of the Web service is created and added to the service registry, otherwise the Web service is rejected as faulty. The benefit of performing the verification procedure at the broker site, as opposed to performing it at the consumer site upon discovery, is that it needs to be done only once. Since only successfully tested Web services are accepted by the broker, consumers are ensured that the Web services they discover have been verified with respect to their specifications.

4.3 Consumer's perspective

As a first step during discovery, the service consumer formulates a service request and submits it to the service registry. In response, the service broker returns a set of annotated service descriptions that match the service request. Notably, our approach is not bound to any particular matchmaking mechanism, so that any existing mechanism may be employed to perform syntactic or semantic matchmaking between the service request and the service advertisements. The service consumer can take advantage of the SXM behavioural model provided with each service candidate, in order to perform service selection. This is a validation process where the consumer ensures that a service model satisfies his or her requirements. An important validation technique is model animation, during which the user feeds the model with sample inputs and observes the current state, transitions, processing functions, memory values, and last but not least, the outputs. For example, X-System is a prolog-based tool supporting the animation of stream X-machine models [10]. In addition, model checking may be employed on the SXM model to check for desirable or undesirable properties, which are specified in a temporal logic formula. Research on X-machines offers a model-checking logic, called XmCTL, which extends Computation Tree Logic (CTL) with memory quantifiers in order to facilitate model-checking of X-machine models [20]. Alternatively, if the consumer has a SXM model of the required service, it can be validated by state and transition refinement against the published SXM of the provided service [21].

5. Conclusions

The approach described in this paper is supported in fragments by a number of existing tools, which have been developed during previous research. However, numerous gaps exist in the required supporting infrastructure, and future research will address the

consolidation of techniques and tools into a single framework with industrial applicability. The main focus will be on the broker infrastructure, which requires more substantial work to support automated Web service testing and, possibly, behavioural matchmaking. We will base our work on a semantically-enhanced, UDDI-based service registry supporting the SAWSDL specification, as part of the EU-funded STREP project FUSION [22]. In order to support model-based testing of Web services, we are planning to integrate the semantic registry with tools for test case generation from XMDL specifications, and with capabilities for runtime testing of a Web service implementation. Additionally, in order to support behavioural matchmaking, we are planning to define an abstract query language for the service consumer and extend the current matchmaking algorithm of the semantic service registry to match the behavioural query with the advertised stream X-machine models.

References

1 Papazoglou, M. P., Traverso, P., Dustdar, S., Leymann, F. (2006). Roadmap of Service Oriented Computing. http://infolab.uvt.nl/pub/papazogloump-2006-96.pdf. March 2006.

2 G. Laycock. The Theory and Practice of Specification-Based Software Testing. PhD thesis, Dept of Computer Science, Sheffield University, UK, 1993.

3 Holcombe, M. and Ipate, F. (1998). Correct Systems: Building Business Process Solutions. Springer-Verlag, Berlin.

4 Ipate, F. and Holcombe, M. (1997). An integration testing method that is proven to find all faults. International Journal of Computer Mathematics, 63, pp 159-178.

5 Sinha, A. and Paradkar, A. (2006). Model-based functional conformance testing of Web services operating on persistent data. In TAV-WEB'06, pages 17{22, Portland, Maine, USA, 2006. ACM.

6 Keum, C., Kang, S., and Ko, I.Y. (2006). Generating test cases for web services using extended finite state machine. In TestCom 2006, pp 103-117. Springer, 2006.

7 Bertolino, A., Frantzen, I., Polini, A., and Tretmans, J. (2006). Audition of web services for testing conformance to open speci_ed protocols. Architecting Systems with Trustworthy Components, LNCS 3938, 2006.

8 Heckel, R. and Mariani, L. (2005). Automatic conformance testing of web services. In FASE 2005, pp 34-48. Springer, 2005.

9 Eilenberg, S. (1974). Automata, languages and machines. Academic Press, New York.

10 Kapeti, E. and Kefalas, P. (1999). A design language and tool for X-machine specification. In Proceedings of the 7th Panhellenic Conference on Information Techology, Greek Computer Society, Ioannina, 1999.

11 Kefalas, P. (2000). X-machine description language: User manual, version 1.6. Technical Report WP-CS07-00, CITY College, 2000.

12 Kefalas, P., Eleftherakis, G., and Sotiriadou, A. (2003). Developing tools for formal methods. In Proceedings of the 9th Panhellenic Conference in Informatics, pages 625-639, November 2003.

13 Dranidis, D., Eleftherakis, G., and Kefalas, P. (2005). Object-based language for generalized state machines. Annals of Mathematics, Computing and Teleinformatics (AMCT), 1(3):8-17, 2005.

14 Dranidis, D., Kourtesis, D. and Ramollari, E. (2007). Formal Verification of Web Service Behavioural Conformance through Testing. In Proceedings. of the 3rd South-East European Workshop on Formal Methods, Thessaloniki, Greece, November 2007.

15 F. Ipate and R. Lefticaru. State-based testing is functional testing. Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007, pages 55-66, 10-14 Sept. 2007.

16 T. S. Chow. Testing software design modelled by finite state machines. IEEE Transactions on Software Engineering, 4:178-187, 1978.

17 E. Ramollari, D. Kourtesis, D. Dranidis, and A.J.H. Simons. Towards Reliable Web Service Discovery through Behavioural Verification and Validation. In Proceedings of the 3rd European Young Researchers Workshop on Service Oriented Computing (YR-SOC 2008), London, UK, June 2008 (forthcoming).

18 D. Kourtesis, E. Ramollari, D. Dranidis, and I. Paraskakis. Discovery and Selection of Certified Web Services through Registry-Based Testing and Verification. In Proceedings of the 9th IFIP Working Conference on Virtual Enterprises (PRO-VE'08), Poznan, Poland, September 2008 (forthcoming).

19 J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema. W3C Candidate Recommendation, January 2007. Available at: http://www.w3.org/TR/sawsdl/.

20 G. Eleftherakis, P. Kefalas, and A. Sotiriadou. Xmctl: Extending temporal logic to facilitate formal verification of x-machines. Matematica-Informatica, 50:79-95, 2002.

21 A. J. H. Simons. A theory of regression testing for behaviourally compatible object types. Software Testing, Verification, and Reliability, 16(3):133-156, August 2006.

22 Kourtesis, D. and Paraskakis, I. (2008). Web service discovery in the FUSION semantic registry. In Proceedings of the 11th International Conference on Business Information Systems (BIS 2008), May 2008.